Model Predictive Feedback Control for QoS Assurance in Webservers

Cheng-Zhong Xu and Bojin Liu, Wayne State University
Jianbin Wei, Yahoo!

The *e*QoS framework represents a first step in a systematic approach to provisioning user-perceived page-view response time differentiation and assurance. It deploys a model predictive feedback controller approach to adjust the processing rate allocated to each class of requests dynamically in response to measured performance deviation.

Internet servers configured based on mean-value analysis of resource demands are prone to forming bottlenecks because of traffic dynamics. Quality-of-service-aware resource management provides coordinated allocation of resources to client requests, ensuring that preferred clients receive quality assurance while guaranteeing others a fair and graceful performance degradation when a server overloads.[1]

In contrast to today's best-effort service model, the QoS differentiation-and-assurance architecture creates an opportunity to implement a differentiated pricing structure in the next generation of Internet services. Even on an indiscriminate website, the architecture can isolate the performance of requests from different clients or proxies. By downgrading the quality of their requests, the architecture controls aggressive clients' behaviors and ensures fair sharing between clients, proxies, and even traffic from different domains. Fairness assurance automatically builds a firewall against aggressive clients and protects the server from flash-crowd-like distributed denial-of-service attacks.

The QoS differentiation and assurance concept is rooted in early studies on QoS-aware packet scheduling and congestion control for Differentiated Services (DiffServ) and Integrated Services (IntServ) in the network core. The network alone cannot provide sufficient application-level QoS differentiation and assurance. An early critical path analysis of TCP transactions revealed that server-side delay constitutes more than 80 percent of client-perceived latency for small requests when server load is high.[2] In the past decade, network overprovisioning has made QoS failure rare in the network core for even large, media-rich requests. Request queuing delays and their prolonged processing time in overloaded servers have become increasingly important factors for end-to-end QoS assurance.

Many studies have focused on QoS-aware resource management for service differentiation in Internet servers. The "QoS-Aware Resource Management Approaches in Webservers" sidebar describes related research efforts. Most of the past work emphasized server-side QoS control with respect to primitive performance metrics like connection and request queuing delay and the response time of individual requests. In practice, both network delays and server response time affect client-perceived service quality. Moreover, a webpage often contains multiple embedded static or dynamic objects. Figure 1 shows the interactions between a client retrieving data from a webserver. Figure 1a shows a webpage example that contains 18 embedded objects, while Figure 1b shows the webserver from which the data is retrieved. User-perceived QoS should be measured by page-view response time instead of a single request or connection in webservers.

Our *end-to-end QoS provisioning framework*, *e*QoS, monitors and controls user-perceived QoS with respect to page-view response time. The delay from when a decision is made to when the effect of control is observed

Published by the IEEE Computer Society

QoS-Aware Resource Management Approaches in Webservers

QoS-aware resource management in webservers has received much attention in the past decade. Early studies took the basic approach of assigning different numbers of virtual processors—threads, processes, or processors in parallel computers—and their priorities to different request classes.[1] Such approaches could provide different QoS levels to different classes, but they are insufficient to control quality spacings and provide any QoS guarantees.

Many researchers tried to apply queuing models to infer expected delays of a traffic class in webservers according to the input load change and to estimate a processing-rate share for the class to bound its delay.[2,3] Urgaonkar and his colleagues[4] developed a queue network model for performance characterization of multitier websites. Because Web traffic has self-similarity, queuing model-based resource allocation approaches have limited accuracy. To deal with this, recent studies were conducted to evaluate the use of feedback control approaches to regulate the processing-rate allocation in response to measured deviations from the desired performance. For example, C. Lu and his colleagues[5] presented a feedback control design for QoS differentiation and assurance in webservers with respect to connection delay. Diao and his colleagues[6] applied feedback control approaches for service differentiation in multitier websites. Y. Lu and her colleagues[7] applied a PI controller to adjust the processing rate estimated based on an M/M/1 system in response to the changed workload input.

J. Wei and his colleagues[8] presented an integrated queuing model with feedback control for proportional slowdown differentiation between classes of traffic in a general M/G/1 queuing system. Slowdown is a normalized queuing delay of a request with respect to its processing time. This approach used queuing model-based predictions to compensate for the effect of prolonged dead time caused by the factor of processing time in the control loop. Our work extends this approach for the provisioning of client-perceived page-view response time guarantees.

Admission control is a complementary approach to processing rate allocation for QoS assurance. X. Chen and P. Mohapatra[9] applied an early random drop approach in network routing for admission control of requests in Internet servers. They showed the effectiveness of their approach in provisioning differentiated services in terms of queuing delays between lower and higher priority classes of traffic but with no guarantee of quality spacings.

T. Abdelzaher and his colleagues[10] treated requests as aperiodic real-time tasks with arbitrary arrival and computation times and relative deadlines. They applied linear feedback control theories to admit an appropriate number of requests to keep system utilization bounded in real-time scheduling. A. Kamra and his colleagues[11] presented a self-tuning PI controller to adjust the admission rate in multitiered websites to guarantee the response times of admitted requests. Their work complements our proposed approach by providing QoS control in terms of system utilization and request response times.

References

1. J. Almeida et al., "Providing Differentiated Levels of Service in Web Content Hosting," *Proc. ACM Sigmetrics Workshop Internet Server Performance*, ACM Press, 1998, pp. 91-102.
2. L. Sha et al., "Queueing Model-Based Network Server Performance Control," *Proc. IEEE Real-Time Systems Symp.* (RTSS), IEEE Press, 2002, pp. 81-90.
3. X. Zhou, J. Wei, and C. Xu, "Resource Allocation for Session-Based 2D Service Differentiation in E-Commerce Servers," *IEEE Trans. Parallel and Distributed Systems*, vol. 17, no. 8, 2006, pp. 838-850.
4. B. Urgaonkar et al., "An Analytical Model for Multi-tier Internet Services and Its Applications," *Proc. ACM Sigmetrics*, ACM Press, 2005, pp. 291-302.
5. C. Lu et al., "Feedback Control Architecture and Design Methodology for Service Delay Guarantees in Web Servers," *IEEE Trans. Parallel and Distributed Systems*, vol. 17, no. 9, 2006, pp. 1014-1027.
6. Y. Diao et al., "Controlling Quality of Service in Multi-tier Web Applications," *Proc. IEEE Int'l Conf. Distributed Systems* (ICDCS), 2006, p.25.
7. Y. Lu et al., "Feedback Control with Queuing-Theoretic Prediction for Relative Delay Guarantees in Web Servers," *Proc. IEEE Real-Time and Embedded Technology and Applications Symp.* (RTAS), IEEE Press, 2003, pp. 208-217.
8. J. Wei, X. Zhou, and C. Xu, "Robust Processing Rate Allocation for Proportional Slowdown Differentiation on Internet Servers," *IEEE Trans. Computers*, vol. 54, no. 8, 2005, pp.964-977.
9. X. Chen and P. Mohapatra, "Performance Evaluation of Service Differentiating Internet Servers," *IEEE Trans. Computers*, vol. 51, no. 11, 2002, pp. 1368-1375.
10. T. Abdelzaher, K. Shin, and N. Bhatti, "Performance Guarantees for Web Server End-Systems: A Control Theoretical Approach," *IEEE Trans. Parallel and Distributed Systems*, vol. 13, no. 1, 2002, pp. 80-96.
11. A. Kamra, V. Misra, and E. Nahum, "Yaksha: A Self-Tuning Controller for Managing the Performance of 3-Tiered Websites," *Proc. Int'l Workshop Quality of Service* (IWQoS), IEEE Press, 2004, pp. 47-56.

presents a challenging issue in the design of a feedback controller.[3] The delay is often referred to as *dead time* or *process delay* in control theory, and a long dead time leads to severe controller instability.

To compensate for dead time, the *e*QoS framework uses a server queuing model to predict how changes made now by the controller will affect the page-view response time in the future. We demonstrate the stability and subsecond control accuracy of the predictive feedback controller on PlanetLab,[4] a worldwide distributed Internet testbed. To the best of our knowledge, this system is the first to provide client-perceived page-view-oriented QoS assurance.

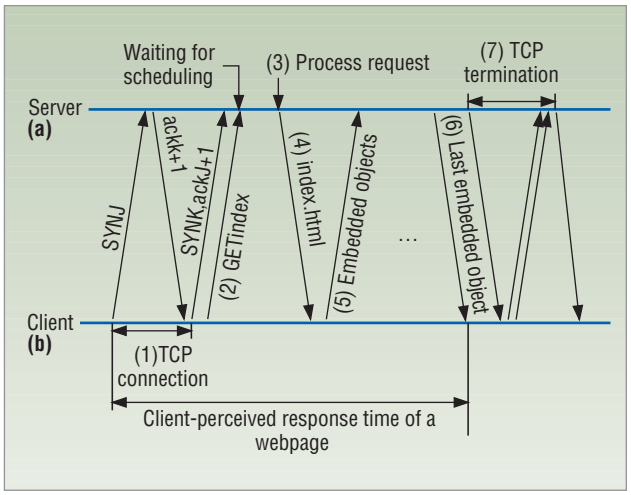

Figure 1. Interactions between (a) the client and (b) the webserver during retrieval of www.wayne.edu: (1) establish a TCP connection, (2) request index.html, (3) process request, (4) return index.html, (5) request embedded objects, (6) return embedded objects, (7) close the TCP connection.

*E*QOS FRAMEWORK

We designed the *e*QoS framework to provide client-perceived end-to-end QoS guarantees in webservers. The framework consists of three key components: a QoS controller, resource manager, and QoS monitor. Figure 2 shows the system architecture and its interface to an Apache webserver.

Page-view response time and QoS monitor

The QoS monitor measures client-perceived service quality in real time, considering both network transfer time and server-side queuing delays and processing time. The client-perceived page-view response time starts when a client sends the first request for the webpage to the server and ends when the client receives the webpage's last object. The processing time equals the period the server spends processing the requests for the webpage and its embedded objects.

Assume that all objects reside on the same website so that we can control the processing. A typical multitier website has a webserver front end. Figure 1b shows the interactions between clients and an Apache server with HTTP/1.1 support during the retrieval of www.wayne.edu/index.html.

Accurately measuring client-perceived page-view response time in real time is crucial to the *e*QoS framework's design. The QoS monitor performs this measurement from the server side similarly to the *ksniff* approach[5]: capturing live network packets and extracting TCP/IP and HTTP header information for reconstruction of accessed webpages. Page-view response time is then derived from the accesses to different webpages. This does not require any client modification. Recent advances in nonintrusive measurement of HTTPS traffic page-view response times[6] made it possible to deploy the *e*QoS framework on secured webservers.

Prior to establishing a TCP connection, the server IP address must be resolved. The latency incurred in this step is unknown to the server. In the range of 10 to 100 ms in normal conditions, this latency is negligible compared with the response times of webpages, which normally span full seconds.

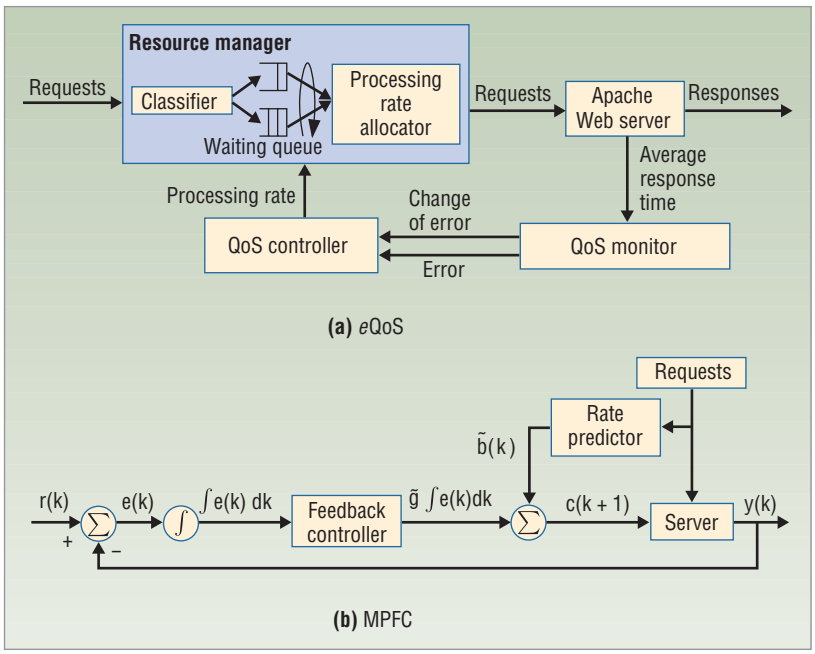

Figure 2. System architecture with Apache webserver interface. (a) e QoS framework; (b) model predictive feedback control.

Resource management

The resource manager classifies and manages client requests and facilitates

resource allocation between classes. It comprises a classifier, classified waiting queues, and a processing-rate allocator. The classifier categorizes requests according to rules defined by service providers. These rules can be based on the request's header or application-level information. The system stores classified requests in their corresponding waiting queues, serving requests from the same class on a first-come, first-served basis. The resource manager serves requests in different queues in a work-conserving, weighted fair-queuing discipline to guarantee fair sharing of the server's processing rate between different classes.

Rate-allocation realization presents a challenge when using rate-based resource-allocation techniques: For a given share, the amount of resources allocated to deliver the expected processing rate must be determined. Our framework treats each request as a scheduling unit and assigns a weight to each class. The resource manager processes requests from all nonempty classes in the order of their weights. Once it dispatches a request, its class weight decrements by one. For Apache servers in a process model, our system implements the rate-allocation process by controlling the number of child processes allocated to each class.

Rate allocation for the provisioning of target QoS guarantees poses another challenge in QoS-aware resource management. Feedback control offers one general approach for processing rate allocation. It operates by adjusting the processing rate share dynamically in response to measured deviations from desired performance. Feedback control stability relies on an accurately measured effect of previous resource allocation on client-perceived performance. This, in turn, controls the order in which the system schedules client requests. The dead time from the decision time of resource allocation adjustment to the observation time of its effect sets a fundamental limit on how well a controller can fulfill design specifications because it limits how fast the controller can react to disturbances.

Apache server interface

We implemented a prototype of the *e*QoS framework as an application-level QoS-aware resource-management module, external to the multitier website, with an Apache server front end. The underlying OS might, however, drop new connection requests to a heavily loaded server due to an accept queue overflow in the kernel. Such dropping will in turn trigger client-side exponential backoff of TCP connection requests.

To avoid uncontrolled retransmission delays, the *e*QoS system drains the kernel's accept queue in a tight loop and stores the accepted connections in the resource manager's different per-class queues. We modified the Apache server to have it accept requests from the resource manager through a Unix domain socket. Whenever the server has an idle child process, it asks the *e*QoS system for another request to handle.

MODEL PREDICTIVE FEEDBACK CONTROL

Many feedback control approaches can readily be applied to QoS-aware resource management. One popular control design technique, proportional-integral-derivative (PID) loops, maintains the output at a target value by taking into account the output error, change of error over a given period, and derivative of the error.[3] Since the *e*QoS framework aims to provide page-view response time guarantees, the target incurs a long dead time, which in turn poses a severe stability problem for the PID controllers.

Our model predictive feedback controller (MPFC) tackles the instability issue, using a server queuing model to predict future system behavior and compensate for the dead-time effect. In general, MPFC approaches improve on a standard feedback loop by predicting how a system would react to input. From the predictive model's perspective, as the effects of inputs will be mostly known ahead of time, feedback control loops can be used to correct model inaccuracies.

Figure 2b shows our MPFC approach's control loop. It comprises two key components: a model-based rate predictor and an integrated feedback controller. The controller uses integral control to adjust the rate allocation according to the difference between the target and achieved performance. To improve the controller's agility, the rate predictor estimates the processing rate of a class according to its predicted workload.

The feedback controller

The target performance can be measured either by an absolute page-view response time guarantee or by a relative quality spacing between different classes.[7] For a design that provides absolute responsive time guarantees, we assume N request classes. Let $T_i(k)$ denote the average page-view response time of class i, $1 \le i \le N$, at sampling period k, $k = 0, 1, 2, \ldots$. An absolute responsiveness guarantee is to ensure for any class i,

$$T_i(k) \le D_i, 1 \le i \le N, \quad (1)$$

where D_i is a predefined page-view response time bound. Let c_i denote the allocated processing rate of class i in a fraction of the server's capacity. The work conservation law requires that the server never be idle if backlogged requests a wait service in the queues. That is,

$$\sum_{i=1}^{N} c_i = 1. \quad (2)$$

This constraint implies that not every set of response time bounds D_i is feasible. Without loss of generality, we assume $D_1 \ge D_2 \ge \ldots \ge D_N$. That is, the system sorts the classes in a nondecreasing order of preference. The feedback controller adjusts the processing rate c_i of each

class to ensure that the maximum number of most preferred classes meet their responsiveness targets. When the server becomes highly loaded, the quality of least preferred classes must be compromised.

We associate a control loop with each class. For the control loop of class i, the reference input $r_i(k) = D_i$, the control output $y_i(k) = T_i(k)$, and the error $e_i(k) = r_i(k) - y_i(k) = D_i - T_i(k)$. When the set of response time bounds D_i is not feasible, the feedback controller sets $e_i(k) = 0$ for the least preferred classes. It means there will be no feedback control of rate allocation for these basic classes. They will take the remaining processing rate in a best-effort manner after satisfying the preferred classes.

The feedback controller of a preferred class i sums up the errors caused by previous rate allocations and adjusts its processing rate as

$$c_i(k) = c_i(k-1) + g \cdot e_i(k), \tag{3}$$

where g is a control gain that determines the adjustment of the processing-rate ratio corresponding to the error. Let b_i denote a model predictive rate based on a queuing model of the server. It follows that the processing rate of the next sampling period $k + 1$ is

$$c_i(k+1) = b_i(k) + \tilde{g} e_i(k), \tag{4}$$

where $\tilde{g} = g \cdot s$ and $s = \partial T_i / \partial c_i$ is a model-predictive scaling factor of the processing rate with respect to the change of output error in the response time.

The control gain affects the feedback controller's agility. A controller with a large control factor can respond quickly to errors. However, it can cause large oscillations as well. To reduce the overshoot the integral control introduces, a small control gain is preferred.

The rate predictor

This tool periodically calculates the processing rate of a class based on a queuing model to improve the controller's agility. We model the server in a queuing system with a per-class Poisson request-arrival process with an arrival rate of λ_i for class i. The request processing time follows a heavy-tailed bounded Pareto distribution, as some past workload characterization studies have pointed out.[8] Let x be a request's service time, which is bounded by p and q and $p \le x \le q$. The probability density function of the bounded Pareto distribution is defined as

$$f(x) = \frac{\alpha p^{\alpha}}{1 - (p/q)^{\alpha}} x^{-\alpha-1} \tag{5}$$

where α is the shape parameter. We refer to this M/G/1 model with a bounded Pareto service time distribution as an $M/G_P/1$ queuing system.

Recall that three factors determine the user-perceived response time $T_i(k)$ of a request in class i: request service time x_i, queuing delay w_i, and request/response round-trip transmission time d_i. There is no accurate model for the round-trip delay between a server and its clients. Since rate allocation on the server has no effect on d_i, we regard it as a random variable, representing the measurement error of processing time x_i. When the server is highly loaded, x_i is much greater than d_i. With model-predictive rate allocation, we seek to estimate a processing rate b_i for each preferred class such that

$$\mathrm{E}[w_i] + \mathrm{E}[x_i] \le D_i. \tag{6}$$

We derived the expected delay and the first- and second-order statistics of processing time[7] as

$$E[x_i] = \begin{cases} \dfrac{h(\alpha)}{b_i h(\alpha-1)} & \text{if } \alpha \neq 1, \\ \dfrac{(\ln p - \ln q) h(\alpha)}{b_i} & \text{if } \alpha = 1; \end{cases} \tag{7}$$

$$E[w_i] = \frac{\lambda_i h(\alpha)}{2h(\alpha-2)(1 - \lambda_i \mathrm{E}[x_i])}, \tag{8}$$

$$E[x_i^2] = \frac{h(\alpha)}{b_i^2 h(\alpha-2)}, \tag{9}$$

where $h(\alpha) = \dfrac{\alpha p^{\alpha}}{1 - (p/q)^{\alpha}}$.

By setting the request-server-side response time to a target D_i, we have that

$$b_i = \frac{\lambda_i Y}{-Z + \sqrt{Z^2 + 2D_i \lambda_i Y}} \tag{10}$$

where $Y = \mathrm{E}[x_i^2] - 2(\mathrm{E}[x_i])^2$ and $Z = 1 + D_i \lambda_i \mathrm{E}[x_i]$. (11)

EXPERIMENT SETTINGS

We have conducted experiments on the PlanetLab testbed to evaluate the performance of the *e*QoS in a real-world environment. The clients reside on nine nodes in three geographically diverse locations: Cambridge, Massachusetts; San Diego, California; and Cambridge, UK. To treat clients with different network connections fairly, we assume they can be from any nodes.

We ran an Apache webserver in Detroit, Michigan, with support for HTTP/1.1 in a Dell PowerEdge 2450 configured with a 1-GHz Pentium III dual processor and 512-Mbyte main memory. Under normal conditions, the average round-trip times between the server and the clients takes about 45 ms to Cambridge, 70 ms to San Diego, and 130 ms to the UK. We used SURGE,[9] a popular Web traffic generator, to generate the server workload. Adjusting the number of concurrent user equivalents in SURGE controlled the workload level. In the emulated Web objects, we set the maximum number

of embedded objects in a given page to 150, and the percentage of base, embedded, and loner objects to 30, 38, and 32 percent, respectively.

FRAMEWORK EVALUATION

We evaluated the effectiveness of the *e*QoS framework under different server loads and network delays in terms of performance deviation from the targeted page-view response time. Recall that $e(k)$ is the deviation of a preferred class at sampling period k, such that $e(k) = T(k) - D(k)$. We define a metric of overall performance deviation, $R(e)$, as

$$R(e) = \frac{\sqrt{\sum_{k=1}^{n} e(k)^2 / n}}{D(k)} \qquad (12)$$

This relative deviation metric characterizes the control loop's stability. The smaller the $R(e)$, the more the achieved average response time concentrates near the target value and the better the controller's performance.

We conducted experiments with inputs of different numbers of user equivalents, ranging from 1,000 to 1,300. When we hosted fewer than 1,000 user equivalents, the server status remained lightly loaded and every request class received a response within 5 seconds. When the server loaded more than 1,300 user equivalents, we observed refused connections, which require admission control to ensure aggregate webserver performance. We grouped the requests into two categories—premium and basic—and set the page-view response time for the premium class to 5 seconds.

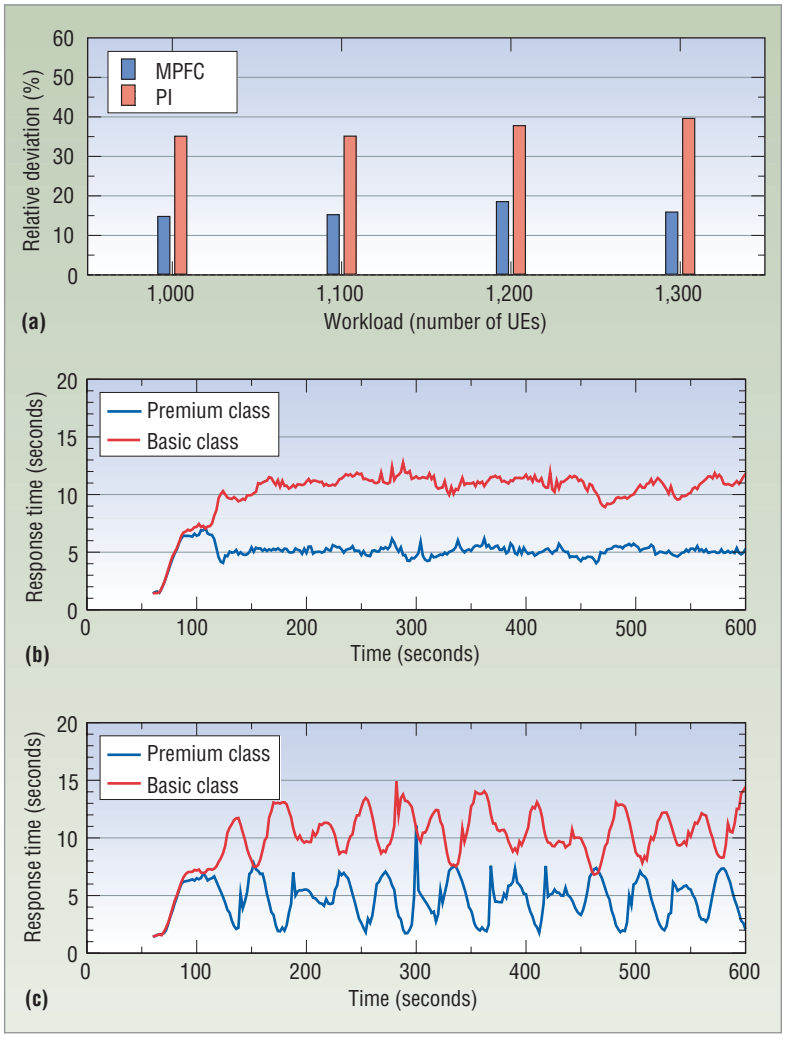

***Figure 3.* eQoS framework performance on PlanetLab. (a) Page-view response time deviation R(e) of the premium class relative to its target 5 seconds, over the period from the 100th until the 600th second; (b, c) actual page-view response times of the premium and basic clients due to the MPFC and PI controllers, respectively.**

In these experiments, to saturate the server we turned on the QoS controller after a 60-second warm-up, then enabled *e*QoS at the 100th second. We set the sampling period's size to 4 seconds.

Figure 3 shows the *e*QoS framework's performance on PlanetLab. Figure 3a shows the page-view response time deviation $R(e)$ of the premium class relative to its target 5 seconds, over the period from the 100th until the 600th second. In this framework, the QoS controller can be implemented in various algorithms. For comparison, the figure also includes the relative deviation results of a standard PI controller.

Although implementing proportional control is simple, it will likely produce steady-state errors. Integral control helps eliminate such errors without overreaction to measurement noises. Recall that in the rate predictor design, we treat a request's transmission time as a measurement error of the request service time. We did not include derivative control in the feedback loop because the controller could become sensitive to measurement errors. Figures 3b and 3c show the actual page-view response times of the premium and basic clients due to the MPFC and PI controllers, respectively.

The figures show that the *e*QoS framework can provide preferred clients with end-to-end response-time guarantees in a long timescale. The feedback controller largely determines the deviation in a small timescale. The figures also show that the PI controller yields a large deviation because of the long dead time in the control loop. In contrast, the model predictive controller can keep the deviation below 20 percent and maintain a subsecond control accuracy. It also keeps the response time tightly bounded even in a small timescale.

We conducted more experiments with different numbers of classes and different round-trip times, achieving results generally consistent with those in Figure 3. We also evaluated the *e*QoS overhead by repeating the experiments in a server with and without *e*QoS. Recall that the resource manager in *e*QoS must copy established connections from the kernel space to the application space for classification and scheduling. It then hands off the connections to the server for processing. Experimental results show that *e*QoS incurs marginal overhead when the server becomes highly loaded.

The *e*QoS framework represents a first step in a systematic approach to the provisioning of user-perceived page-view response-time differentiation and assurance. It deploys an MPFC approach to adjust the processing rate allocated to each class of requests dynamically, in response to measured performance deviation. To compensate for the effect of dead time and enhance control-loop stability, the approach uses an $M/G_P/1$ server-queuing model to predict the future effect of a current processing rate change on page-view response time.

Although the MPFC approach combines the strength of a queuing model with the flexibility of feedback control, it is not immune to limitations in model accuracy. While a bounded Pareto distribution characterizes request service time well, its shape parameter cannot be determined without extensive experimental testing in the server. This limits the portability of the MPFC approach for input traffic with time-varying characteristics and among servers with different capacity and configurations. In such highly dynamic environments, the *e*QoS framework can use a model-free self-tuning feedback control approach in support of adaptive QoS assurance.[10]

Finally, the *e*QoS framework is not limited to QoS-aware resource management in single-tier webservers. It is applicable to multitier websites because the processing time of a request in the entry tier normally involves a sequence of synchronous operations in downstream tiers. However, a multitier website's internal structure might lead to a further increase in the control loop's dead time—a situation that calls for new dead-time compensation techniques.

Acknowledgments

We thank Song Fu, Sithanshu Shashidhara, Haiying Shen, Le Yi Wang, Xiliang Zhong, and Xiabo Zhou for their valuable contributions to the design of the *e*QoS system. This research was supported in part by US National Science Foundation grants ACI-0203592, CCF-0611750, MCS-0624849, CNS 0702488, and CRI-0708232, and by NASA grant 03-OBPR-01-0049.

References

1. C. Xu, *Scalable and Secure Internet Services and Architecture*, Chapman & Hall/CRC, 2005.
2. P. Barford and M. Crovella, "Critical Path Analysis of TCP Transactions," *IEEE/ACM Trans. Networking*, vol. 9, no. 3, 2001, pp. 238-248.
3. G. Franklin, J. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, 4th ed., Prentice Hall, 2002.
4. "PlanetLab: An Open Platform for Developing, Deploying, and Accessing Planetary-Scale Services;" http://www.planet-lab.org.
5. D. Olshefski, J. Nieh, and E. Nahum, "ksniffer: Determining the Remote Client Perceived Response Time from Live Packet Streams," *Proc. Usenix OSDI*, Usenix, 2004, pp. 333-346.
6. J. Wei and C. Xu, "sMonitor: A Non-Intrusive Client-Perceived End-to-End Performance Monitor of Secured Internet Services," *Proc. Usenix Ann. Technical Conf.*, Usenix, 2006, pp. 243-248.
7. J. Wei, X. Zhou, and C. Xu, "Robust Processing Rate Allocation for Proportional Slowdown Differentiation on Internet Servers," *IEEE Trans. Computers,* vol. 54, no. 8, 2005, pp. 964-977.
8. M. Harchol-Balter and A. Downey, "Exploiting Process Lifetime Distributions for Dynamic Load Balancing," *ACM Trans. Computer Systems*, vol. 15, no. 3, 1997, pp. 253-285.
9. P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," *Proc. ACM Sigmetrics 98*, ACM Press, 1998, pp. 151-160.
10. J. Wei and C. Xu, "eQoS: Provisioning of Client-Perceived End-to-End QoS Guarantees in Web Servers," *IEEE Trans. Computers*, vol. 55, no. 12, 2006, pp.1543-1556.

Cheng-Zhong Xu *is a professor in the Department of Electrical and Computer Engineering at Wayne State University. His research interests include networked computer systems, mobile and embedded computing, and Internet technology. Xu received a PhD in computer science from the University of Hong Kong. He is a senior member of the IEEE and a member of the ACM. Contact him at czxu@wayne.edu.*

Jianbin Wei *is a technical Yahoo! at Yahoo!. His research interests include computer systems and networks, in particular system self-management and optimization. Wei received a PhD in computer engineering from Wayne State University. Contact him at jianbin.wei@sdsmt.edu.*

Bojin Liu *is a graduate student in the Department of Electrical and Computer Engineering at Wayne State University. His research interests include Internet service quality control, P2P, and distributed storage systems. Liu received a BS in computing from Hong Kong Polytechnic University. Contact him at fredlbj@wayne.edu.*